Esercizio di Sincronizzazione tra Processi: L'Ascensore

Testo:

Un palazzo di piani PIANI ha un ascensore capace di accogliere un numero infinito di passeggeri.
Ogni passeggero richiede di essere trasportato da un piano all'altro usufruendo dell'ascensore.
Per ottenere una politica di servizio senza penalizzazione l'ascensore tende a mantenere il proprio verso di movimento (verso l'alto o verso il basso), fino ad arrivare a fine corsa (politica di movimento a spazzola).
I passeggeri specificano un piano di partenza e uno di arrivo. Devono ovviamente sospendersi se l'ascensore non è presente al piano di partenza o non va nel verso corretto. I passeggeri scendono dall'ascensore quando questo arriva al piano di arrivo.

Determinare la politica di sincronizzazione, spiegarla adeguatamente e quindi realizzarla con i costrutti monitor e facoltativamente regioni critiche condizionali.

Soluzione con il Costrutto Monitor:

{ Definiamo la politica dell' ascensore nel modo più semplice possibile per evitare la starvation: l'ascensore spazzola dal piano superiore al piano inferiore e viceversa, anche se non sono presenti richieste per i piani verso cui si sta muovendo.
Politiche più sofisticate od efficienti sono ovviamente possibili.

Consideriamo un processo per l'ascensore ed un processo per ogni singolo utente. Il monitor disciplina gli accessi alla struttura che rappresenta lo stato corrente della risorsa ascensore, cioè il piano e la direzione di movimento.}

```
program ascensoreSenzaLimitiDiCapacita;
const N = ..;
type piano = 1 .. N;
    dir  = ( su, giu );

type ascensore = monitor ;
var  pianocor : piano;
    dircor   : dir;
    csu, cgiu, carr : array [ piano ] of condition;
{definiamo tra condizioni per ogni piano:
le prime due sono per l'accesso all'ascensore nei due versi;
l'altra per l'arrivo al piano.
Le condizioni consentono di sincronizzazre l'utente con il movimento dell'ascensore }

          procedure entry mov ( part : piano; arrivo : piano);
var  mydir : dir;
begin
    if arrivo > part then mydir := su else mydir := giu;
    if arrivo <> part then
    begin
    { accodamento per l'ingresso all'ascensore }
        if (part <> pianocor) or (mydir <> dircor) then
            if mydir = su then csu  [part]. wait
            else              cgiu [part].wait;
```

```
    { l' utente deve essere accodato al piano di arrivo }
        carr [ arrivo]. wait;
    end;
end;

procedure entry spostaAscensore;
begin
    if dircor = su then
        while csu [ pianocor] .queue do csu [pianocor].signal;
{ per ogni piano segnala tutti i processi sospesi in coda in attesa di entrare nell'ascensore, se la direzione e' corretta }
    else while cgiu [ pianocor] .queue do cgiu [pianocor].signal;

    while carr [pianocor] . queue  do carr [pianocor]. signal;
            { per ogni piano segnala tutti i processi sospesi in coda in attesa di arrivare }
    if dircor=su then
        if pianocor <> N  then pianocor + :=1
            else            dircor := giu
    else if pianocor <> 1  then pianocor - := 1
            else            dircor := su ;
end;
begin dircor := su; pianocor := 1;
end;
type utente = process ;
var piano1, piano2 : piano;
begin
    repeat
        < scegli direzioni >
        elevator. mov ( piano1, piano2);
    until false;
end;

type asc = process;
begin repeat
        elevator. spostaAscensore;
    until false;
end;

var  elevator : ascensore;
    muoviasc : asc;
    u1, ...  : utente;

begin .. end.


{Si possono prevedere gestioni ottimizzate del movimento      dell'ascensore: per esempio di 'spazzolare' fermandosi solo            sui piani per i quali c'e' almeno una richiesta.

 Questa ottimizzazione della politica di gestione rimane confinata all'interno della procedura spostaAscensore }
```